

LEVEL

September 1981

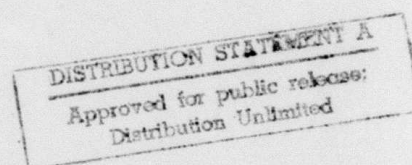
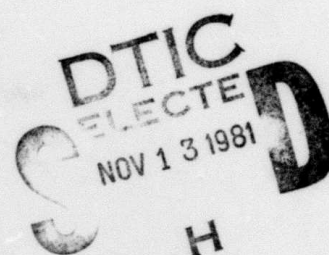
LIDS-P-1149

AD A107462

A DISTRIBUTED ALGORITHM FOR MINIMUM WEIGHT
DIRECTED SPANNING TREES*

by

Pierre A. Humblet**



DTIC FILE COPY

*The author is with the Dept. of Electrical Engineering and Computer Science and the Laboratory for Information and Decision Systems at Mass. Institute of Technology, Cambridge, Mass. 02139.

**This research was conducted at the M.I.T. Laboratory for Information and Decision Systems with partial support provided by the Defense Advanced Projects Agency under Contract ONR-N00014- 75-C-1183 and the National Science Foundation under Contract NSF-ECS 79-19880.

81 10 26 096

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
	AD-A107462	
4. TITLE (and Subtitle)	5. TYPE OF REPORT & PERIOD COVERED	
6. A DISTRIBUTED ALGORITHM FOR MINIMUM WEIGHT DIRECTED SPANNING TREES.	9. Technical rept.	
7. AUTHOR(s)	6. PERFORMING ORG. REPORT NUMBER	
10. Pierre A. Humblet	14. LIDS-P-1149	
9. PERFORMING ORGANIZATION NAME AND ADDRESS	8. CONTRACT OR GRANT NUMBER(s)	
Massachusetts Institute of Technology Laboratory for Information and Decision Systems Cambridge, Massachusetts 02139	✓ ARPA Order No. 3045/5-7-75 ONR/N00014-75-C-1183	
11. CONTROLLING OFFICE NAME AND ADDRESS	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209	Program Code No. 5110 ONR Identifying No. 049-383	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	12. REPORT DATE	
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	11. September 1981	
16. DISTRIBUTION STATEMENT (of this Report)	13. NUMBER OF PAGES	
Approved for public release; distribution unlimited.	21	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)	15. SECURITY CLASS. (of this report)	
	UNCLASSIFIED	
18. SUPPLEMENTARY NOTES	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Distributed Algorithms Computer Networks Spanning Trees		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		
<p>A distributed algorithm is presented to construct minimum weight directed spanning trees (arborescences), each with a distinct root node, in a strongly connected directed graph.</p> <p>A processor exists at each node. Given the weights and origins of the edges incoming to its node, a processor follows the algorithm and exchanges messages with its neighbors until all arborescences are constructed. The amount of information exchanged and the time to completion are $O(N ^2)$.</p>		

410750 LB

572

Abstract

A distributed algorithm is presented to construct minimum weight directed spanning trees (arborescences), each with a distinct root node, in a strongly connected directed graph

A processor exists at each node. Given the weights and origins of the edges incoming to its node, a processor follows the algorithm and exchanges messages with its neighbors until all arborescences are constructed. The amount of information exchanged and the time to completion are $O(|N|^2)$.

1. Introduction

We consider a strongly connected directed graph consisting of a finite set N of nodes and a set $E \subset N \times N$ of edges with a finite weight $w(e)$ assigned to each edge e . We assume that the nodes have distinct identities that are ordered.

The node identity and the weights and origins of all edges incoming to a node are initially given to a processor located at that node. Each processor performs the same local algorithm, which consists of sending messages over adjacent edges, waiting for incoming messages and processing them.

Messages can be transmitted independently in both directions on a directed edge, and arrive after an unpredictable but finite delay, without error and in sequence.

Accession for	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
NTIS Grant			
DTIC Tech			
Unclassified			
Justification			
By			
Distribution/			
Availability Codes			
Avail and/or			
Dist			

A

After a node completes its local algorithm, it knows which adjacent edges (incoming or outgoing) are part of a minimum weight directed spanning tree (arborescence) rooted at each node.

Having a tree with edges directed away from the root is useful in communication networks when one wishes to broadcast information from a node to other nodes in the network. Trees with edges directed toward the root have been proposed for use in distributed database systems [8]. When the topology of the network can change due to failures or additions of links or nodes, it is desirable to be able to build the arborescences in a distributed manner, without having to rely on a central node that can be inaccessible. Dalal and Metcalfe [4] [3] have described a number of distributed algorithms to construct arborescences.

If there is a cost associated with the use of a link in the network, it is useful to determine minimum cost arborescences. This is the object of this paper. An interesting result, besides the algorithm itself, is that the amount of communication between the nodes to find the $|N|$ optimal arborescences is $O(|N|^2)$, which is the same order of magnitude as what it takes to construct any $|N|$ arborescences. The time to complete the algorithm is also $O(|N|^2)$.

If the network graph is not directed, then the problem simplifies to finding a minimum weight spanning tree. Distributed algorithms to that effect have been given by Spira [9] and Gallager, Humblet and Spira [6].

The paper continues with a review of the centralized algorithm to find minimum cost arborescences. It is then explained how the functions can be distributed. The communication cost and running time analysis follow. A precise description of the algorithm appears in appendix.

2. Review of minimum cost arborescences

We assume the reader is familiar with the elementary definitions and properties of graphs, paths, cycles, trees, etc. which can be found for example in [7]. In particular a graph is strongly connected if for every pair of nodes there is a directed path with the first node as origin and the second as destination. An arborescence rooted at a node is a directed tree such that one edge in the tree is incoming to each node, except the root. The weight of an arborescence is the sum of the weights of the edges it includes.

Our objective is to find $|N|$ minimum weight arborescences, one rooted at each node. Clearly this is possible if and only if the graph is strongly connected.

A centralized algorithm to that effect has first been described by Chu and Lin [2], and rediscovered by others [5], [1] using different methods. Tarjan [10] gives an efficient implementation. The algorithm is also described in [7]. We review it briefly in this section. It rests on three observations:

- 1) By definition, any arborescence rooted at a given node contains one and only one edge incoming to every other node. Thus if a constant is added to the weights of all edges

incoming to a node, the weights of the arborescences change by the same amount and minimum weight arborescences before the change remain so after the change. Thus we can, and from now on will, assume that a minimum weight edge incoming to each node has zero weight.

- 2) If a set L_e of zero weight edges form a directed cycle, with L_n denoting the set of nodes in the cycle, then for any node r there is a minimum weight arborescence rooted at r such that all edges in L_e , except one, are in the arborescence. The edges in the arborescence but not in L_e form a minimum weight arborescence for the reduced graph obtained by merging all nodes in L_n into a single node; if r is in L_n , the new arborescence is rooted at this new node instead of at r .

This observation is proved by starting with any optimal arborescence rooted at r , finding the first node f in L_n on a directed path (in the arborescence) from r to any node in L_n , removing from the arborescence all edges incoming to nodes in $L_n \setminus \{f\}$, and adding all edges in L_e , except the one incoming to f (\setminus denotes set subtraction). The result is a new arborescence satisfying the description in the paragraph above. It is optimal as all added edges have zero weight, and all removed edges have non negative weight. The edges in the arborescence but not in L_e form an arborescence for the reduced graph, with same weight as the original arborescence. If the smaller arborescence had not minimum weight, the original arborescence would not either.

- 3) Let A be a set of edges consisting of one zero weight edge incoming to each node. A contains a directed cycle, as a traveler starting at any node and walking in reverse direction on the edges in A will always be able to do so, and will eventually visit the same node twice, the graph being finite.

These three observations suggest the following recursive algorithm to find minimum weight arborescences. For each node, add a constant to the weights of the incoming edges, so that their minimum weight becomes zero. Select enough zero weight edges to form a directed cycle (its existence is guaranteed by observation 3). Let L_e and L_n be the sets of edges and nodes in the cycle. For every edge e in L_e , incoming to node $d(e)$ say, mark e as being on the arborescences rooted at the nodes in $L_n \setminus \{d(e)\}$.

By observation 2, the other edges of the arborescences can be determined recursively by considering the reduced graph obtained by replacing all nodes in L_n by a single node (called a cluster).

The general step of the algorithm is as follows. Start with a graph whose nodes are clusters of nodes. For each cluster subtract a constant from the weights of the edges incoming to the cluster from nodes outside, so that their minimum weight becomes zero. Select enough zero weight edges to form a directed cycle of clusters. Let L_e and L_c be the sets of edges and clusters in the cycle.

For each edge e in L_e , incoming to cluster $c(e)$ say, mark e and the edges between nodes of $c(e)$ already marked as belonging to the arborescence rooted at $d(e)$ as belonging to the arborescences rooted at all nodes included in clusters in $L_c \setminus \{c(e)\}$.

Replace all nodes included in clusters in L_c by a single cluster and repeat the procedure until only one cluster remains.

Note that NRG , the number of reduced graphs produced by the algorithm, lies between one and $|N|-1$. The upper bound results from the fact that a cycle L_e will give rise to a reduced graph with $|L_e|-1 \geq 1$ fewer nodes; the bound can be achieved if all cycles contain two edges (e.g figure 1). The total number of edges that ever become part of a cycle is equal to $|N| + NRG - 1$, as one incoming edge is selected for every node and every cluster, except the last one.

3. Description of the distributed algorithm

A precise description of the distributed algorithm appears in appendix. We relate here how the main functions of the centralized algorithm, i.e. detection of cycles, updating of the arborescences and selection of a minimum weight cluster incoming edge can be distributed. We first describe the data structure maintained by the nodes.

As in the centralized algorithm, each node is part of a cluster, which initially contains only the node itself. A node knows to which node in the cluster (the Root) the minimum weight cluster incoming edge is adjacent. It also knows the identity of the cluster (Cluster_ID), defined as the largest node identity in the cluster.

In the course of the algorithm edges will be selected. The set of all nodes that have a directed path of selected edges to a given node is called the Known_set of that node. A node will also decide that some of its adjacent edges belong to minimum weight arborescences. Inc_edge[n] denotes the incoming edge belonging to the arborescence rooted at node n, while Out_set[n] denotes the set of outgoing edges belonging to that arborescence.

The cycles are detected as follows. All nodes are initially considered to be asleep. In response to a command from a higher level procedure with which we are not concerned here, or when receiving a message from a neighbor, any number of nodes can wake up. A node waking up initializes the Known_set as containing only itself and sets itself as Root, selects

a minimum weight incident edge and sends the message `CONNECT(Known_set)` on that edge.

When a node receives a `CONNECT(Set)` message on edge `l`, it sets `Neighbor_set[l]` to `Set` and also sends on edge `l` the message `LIST(Known_set\Set)`. It includes `l` as belonging to `Out_set[Root]`.

A node receiving `LIST(Set)` transmits the message `List(Set\Neighbor_set[e])` on `e` for all `e` in `Out_set[Root]` and updates `Known_set` to `Known_set U Set`. Thus if a node identity was included in a `CONNECT` message sent on a link then it is not included in the `LIST` message sent in response nor in any `LIST` message transmitted on that link.

One sees further that `LIST` messages are sent only on selected minimum weight incident edges. As shown before a cycle of such edges must exist. It can be detected when a node is received in a `LIST` message that was received previously in a `CONNECT` message, i.e. when `Known_set` \supset `Neighbor_set[e]` for some `e`. Notice that all nodes in the cycle will detect the cycle, and that the `Known_set`'s of all nodes in the cycle contain precisely the identities of all nodes in the cycle, as by assumption the message sent on a link are received in the same order. It can also be observed that if a node identity `n` was included in a `LIST` message transmitted on an edge `e`, then `e` is part of the arborescence rooted at `n`. Thus the `Inc_edge`'s and `Out_set`'s can be updated as `LIST` messages are received and transmitted.

Now that a cycle, and thus a new cluster, is identified the nodes must collaborate to find the minimum weight cluster incoming edge. It can be found as follows. Consider an agreed upon node, e.g. the node with the largest identity in the cluster. Have the node that received the CONNECT message from the selected node send REPORT(Root,Weight) on its previous best incoming edge, where Weight is the weight of its minimum weight edge incoming from outside the cluster. When a node receives a REPORT(Node,Weight) message it combines this new information with its local knowledge to determine the weight and destination of the best cluster incoming edge it knows about and so informs the next node in the cycle by sending REPORT(Node,Weight) on its previous best incoming edge. Eventually the selected node will find the minimum weight cluster incoming edge and new root node. It communicates this information in an UPDATE(Newroot,Weight) message transmitted on the arborescence rooted at Newroot. All nodes subtract Weight from the weights of their incoming edges, and the new Root node sends CONNECT(Known_set) on its new selected incoming edge. A LIST message will be received in answer and propagated by the Root node inside the cluster and beyond. Note that this message will never precede the UPDATE message announcing the identity of the new Root, as both are broadcast on the same arborescence in the cluster.

Note that many cycles can be formed concurrently, but that at a given time a node can only participate to the formation of a single cycle, as it has selected a single best incoming edge. For example in figure 2 the cycles {1,2,3,4} and {5,6,7} can be formed simultaneously, but the bigger cycle {8,{1,2,3,4},9,{5,6,7},10} can only be formed after the two smaller

cycles.

We now explain how to handle cycles containing clusters of nodes. The procedure to find cycles outlined above still works if the LIST messages are transmitted from the cluster root to all nodes in the cluster on the arborescence rooted at the cluster root.

Once a cycle is detected, all nodes in the new cluster must be informed and this requires a special message, the CYCLE message. Note that the detection of a cycle is always done at a neighbor of a cluster root (with the neighbor not a part of the cluster). The neighbor sends a CYCLE message to the root which retransmits it on its arborescence throughout its cluster, but not outside, contrary to the LIST messages. Thus CYCLE messages are only transmitted on edges belonging to `Internal_set`, i.e. the set of edges joining two nodes in the same cluster.

The updating of `Inc_edge` and `Out_set` can still be done as explained above.

The determination of the minimum weight cluster incoming edge is a generalization of the procedure outlined previously, with the transmission of the REPORT taking place on the arborescence of the root of the cluster with largest `Cluster_ID` in the cycle, which becomes the "selected node". The nodes wait until they have received a CYCLE message and also REPORT messages on all edges on which CYCLE had been sent except the edge incoming to the selected node (those edges form the part of the

arborescence of the selected node internal to the new cluster). This is implemented by having the variable `Wait_count` set to the number of such edges plus one, and decrementing it every time a `CYCLE` or `REPORT` is received, until it reaches 0. At this point a node determines the weight and the destination of the best cluster incoming edge it knows about, and sends the message `REPORT(Node,Weight)` on the edge the `CYCLE` message came on, thus toward the selected node which eventually determines the new cluster root and informs the other node by an `UPDATE` message.

The algorithm continues as explained before and it terminates when the weight carried in the `UPDATE` message is ∞ , indicating that there are no more cluster incoming edges.

4. Communication cost analysis

In this section we compute the amount of communication that takes place between the nodes during the course of the algorithm and we compare it with the communication cost of other algorithms.

Note that the messages `CYCLE`, `REPORT` and `UPDATE` have constant lengths, while the messages `CONNECT` and `LIST` have variable lengths, as they include a Set. We will first evaluate the amount of information carried in these two messages.

Every time a node identity is included in a `LIST` message transmitted on an edge, the edge becomes part of the corresponding arborescence. Thus the total number of node identities transmitted in `LIST` messages is $|N|^2$

- $|N|$, and this is also an upper bound on the number of LIST messages.

A node identity is also transmitted in CONNECT messages on all edges that are part of a cycle, but not part of the corresponding arborescence. As seen above, the number of such edges is precisely NRG (between 1 and $|N|-1$), thus the total number of node identities transmitted in CONNECT messages lies between $|N|$ and $|N|^2 - |N|$. The number of CONNECT messages is equal to the number of edges that are part of cycles, thus between $|N|$ and $2(|N|-1)$.

Every time a cluster is formed, every node in the cluster receives a CYCLE message. All nodes except one transmit a REPORT message and receive an UPDATE message. Thus the maximum number $F(|N|)$ of such three types of messages in a network of $|N|$ nodes satisfies the recursive relation

$$F(|N|) \leq \sum_{i=1}^c F(|N_i|) + 3|N| - 2 \quad |N| > 1 \quad (*)$$

where c is the number of clusters forming the final cluster and N_i are the sets of nodes in these clusters. Note that

$$\sum_{i=1}^c |N_i| = |N|, \quad c > 1 \quad \text{and} \quad |N_i| \geq 1 \quad \text{for} \quad 0 < i \leq c \quad (**)$$

By induction on $|N|$ (starting with $F(1)=0$) one can see that the tightest $F(|N|) = .5(|N|-1)(3|N|+2)$. The proof relies on the fact that this $F(\cdot)$ is convex U , thus the maximum of the right hand side of $(*)$, subject to the convex constraints $(**)$, must occur at an extreme point. In fact it

occurs at the point $c=2$, $|N_1|=1$, $|N_2|=|N|-1$. The bound is in fact an equality for a graph like that in figure 1.

One can thus conclude that the communication cost of the algorithm is $O(|N|^2)$, whether one takes as unit the transmission of a node identity or an edge weight, or the transmission of a message. This is remarkably low. Note that any algorithm to construct $|N|$ non necessary optimal arborescences has a communication cost of at least $|N|(|N|-1)$, as every node must be made aware of every other node.

Consider also the two following simple algorithms to construct arborescences. The first one, resulting in non necessary minimal arborescences, is as follows: every node broadcasts its identity on all its outgoing edges, and rebroadcast an identity received from a neighbor on all its outgoing edges the first time it hears about that identity. This way all nodes receive all other identities once on all incoming edges, and the set of edges over which node i 's identity was received for the first time forms an arborescence rooted at i . Notifying the origin of an edge that the edge belongs to the arborescence can be done by sending messages backwards. The communication cost of this simple algorithm is already $O(|E||N|)$!

Another method to construct optimal arborescences involves informing all nodes of the network topology, and let the nodes perform individually the centralized algorithm. Broadcasting the topology to all nodes requires a communication cost of $O(|E|^2)$ or $O(|E||N|)$. The first number

is when the broadcasting is done by "flooding" the network, the second case is when the transmissions are done on spanning trees (that must be built somehow).

A drawback of the algorithm presented here compared to the two other algorithms is that it takes longer to run. Assuming that processing is instantaneous but that transmitting a message requires one unit of time, our algorithm takes $O(|N|^2)$ in the worst case, whereas the two others take $O(|N|)$.

I. Appendix

In this appendix we give a precise description of the algorithm in an ALGOL-like notation. We allow variables to be sets and we have the usual operations on sets. A statement "For $e := \langle \text{Set} \rangle$ do ..." means "For all e in Set do ...", while $\text{Max}(\text{Set})$ is the largest element of Set and $\text{Card}(\text{Set})$ is the number of elements in Set.

The procedure Send, which is not detailed here, causes the message specified as its first argument to be sent on the edge specified as second argument.

We assume that when a message is received it is placed in a first in first out queue, together with the identity of the edge it was received on. While the queue is not empty the processor takes a message from the queue and calls the corresponding procedure. The last argument of the procedure is the edge over which the message was received. When the queue

is empty, the processor waits until a message arrives.

Initially all processors are waiting. On request from a higher level process or when receiving a "wake-up" message from a neighbor (these can take many forms and are not detailed here), the processors execute the procedure WAKE-UP described below. No message generated by the algorithm can be processed before WAKE-UP has been executed.

The set "Incoming" is assumed to initially contain all edges incoming to the node, the arrays "w" and "Origin" must be set to the weights and origins of those edges, Node_ID must be equal to the identity of the node and Node_set is the set of nodes in the graph. All free variables are shared by all procedures.

```

Procedure WAKE_UP()
begin
  Known_set := {Node_ID};
  New_internal_set := nil;
  for n := <Node_set> do In_edge[n] := Out_set[n] := nil;
  Min_weight := oo;
  for e = <Incoming> do if w[e] < Min_weight then
    begin
      Min_weight := w[e];
      Best_edge := e
    end;
  UPDATE(Node_ID, Min_weight, nil)
end;

Procedure CONNECT(Set, l)
begin
  Neighbor_set[l] := Set;
  CHECK_CYCLE(Known_set, l)
end;

Procedure CHECK_CYCLE(Set, l)
begin
  Send_set := Set \ Neighbor_set[l];
  if Send_set ≠ nil then
    begin

```

```

for n := <Send-set> do Out_set[n] := Out_set[n] U {1};
Send(LIST(Send-set),1)
end;
if Neighbor_set[1]  $\subset$  Known_set then
  if Max(Known_set)  $\neq$  Max(Neighbor_set[1]) then
    Wait_count := Wait_count + 1;
  New_internal_set := New_internal_set U {1};
  Send(CYCLE(),1)
  end
end;

```

```

Procedure LIST(Set,1)
begin
  Known_set := Known_set U Set;
  for n := <Set> do Inc_edge[n] := 1;
  for e := <Out_set[Root]> do
    if e  $\in$  Internal_set then
      begin
        for n := <set> do Out_set[n] := Out_set[n] U {e};
        Send (LIST(Set),e)
        end
      else CHECK_CYCLE(Set,e)
    end;
  end;

```

```

Procedure CYCLE(1)
begin
  Root_edge := 1;
  for e := <Incoming> do
    if Origin[e]  $\notin$  Known_set and w[e]  $\leq$  Min_weight then
      begin
        Min_weight := w[e];
        Best_edge := e;
        Best_node := Node_ID
      end;
  for e := <Out_set[Root]  $\cap$  Internal_set> do Send(CYCLE(),e);
  REPORT(Best_node,Min_weight,nil)
end;

```

```

Procedure REPORT(Node,Weight,1)
begin
  if Weight < Min_weight then begin
    Min_weight := Weight;
    Best_node := Node
  end;
  Wait_count := Wait_count-1;
  if Wait_count = 0 then
    begin
      if Node_ID = Root and Cluster_ID = Max(Known_set)
      then UPDATE(Best_node,Min_weight,nil)
      else Send(REPORT(Best_node,Min_weight),Root_edge)
    end
  end

```

end;

```

Procedure UPDATE(Newroot,Weight,1)
begin
If Weight = oo then STOP;
Root := Newroot;
Cluster_ID := Max(Known_set);
Min_weight := oo;
Internal_set := New_internal_set;
Wait_count := Card(Out_set[Root] / Internal_set) + 1;
for e := <Incoming> do w[e] := w[e]-Weight;
for e := <(Out_set[Root] U {Inc_edge[Root]}) \ {1}> do
    Send(UPDATE(Root,Weight),e);
if Root = Node_ID then Send(CONNECT(Known_set),Best_edge)
end

```

Minor improvements can be made. We mention the fact that the number of types of messages can be reduced, e.g. CYCLE() can be replaced by LIST(nil). Moreover if this convention is adopted, message types can be left out entirely, there being enough context information to determine the message types ! The algorithm can also be made to run faster. For example in procedure CHECK_CYCLE the CYCLE message can be sent on link 1 as soon as $\text{Neighbor_set}[1] \cap \text{Known_set} \neq \text{nil}$, whereas it can be sent on all edges in $\text{Out_set}[\text{Root}] \cap \text{Internal_Set}$ when $\text{Neighbor_set}[1] \subset \text{Known_set}$. However care must be taken not to send multiple CYCLE messages on a link.

REFERENCES

1. F. Bock. An algorithm to construct a minimum directed spanning tree in a directed network. In Developments in operations research, Gordon and Breach, New York, 1971, pp. 29-44.
2. Y.J. Chu and T.H. Liu. "On the shortest arborescence of a directed graph." sci.sinica 14 (1965), 1396-1400.
3. Yogen K. Dalal and Robert M. Metcalfe. "Reverse path forwarding of broadcast packets." Communications of the acm 21, 12 (december 1978), 1040-1048.
4. Yogen K. Dalal. Broadcast protocols in packet switched computer networks. Tech. Rept. 128, Digital Systems laboratory, Stanford University, April, 1977. (Ph.D thesis)
5. J. Edmonds. "Optimum branchings." journal of research of the national bureau of standards 71b, (1967).
6. R.G. Gallager, P.A. Humblet and P.M. Spira. A distributed algorithm for minimum weight spanning trees. Tech. Rept. LIDS-P-906A, LIDS-MIT, October, 1979. submitted for publication.
7. Eugene L. Lawler. Combinatorial optimization: Networks and Matroids. Holt, Rinehart and Winston, New York, 1976.
8. Victor Li. Distributed Databases ... Tech. Rept. , MIT Lids, January, 1981.
9. P.M. Spira. "Communication Complexity of Distributed Minimum Spanning Tree Algorithms." Proceedings 2nd Berkeley Conf. on Distributed Data Management and Computer Networks , (June 1977), .
10. R.E. Tarjan. "Finding optimum branchings." networks 7 (1977), 25-35.



Figure 1: Example of worst case.

The cycles are successively $\{1,2\}$, $\{\{1,2\}, 3\}$, $\{\{\{1,2\}, 3\}4\}$,...

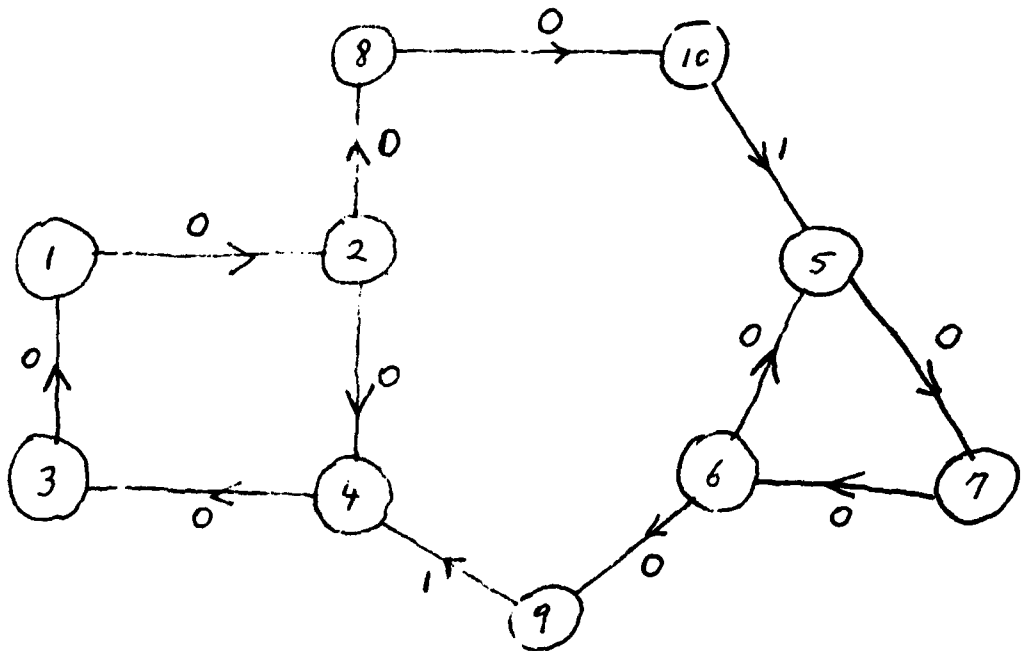


Figure 2

Cycles $\{1,2,3,4\}$ and $\{5,6,7\}$ can be formed simultaneously. Cycle $\{8, \{1,2,3,4\}, 9, \{5,6,7\}, 10\}$ must be formed after the two smaller cycles.

Distribution List

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 Copies
Assistant Chief for Technology Office of Naval Research, Code 200 Arlington, Virginia 22217	1 Copy
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 Copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 Copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois 60605	1 Copy
Office of Naval Research Branch Office, Pasadena 1030 East Greet Street Pasadena, California 91106	1 Copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 Copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 Copy

Office of Naval Research
Code 455
Arlington, Virginia 22217
1 Copy

Office of Naval Research
Code 458
Arlington, Virginia 22217
1 Copy

Naval Electronics Laboratory Center
Advanced Software Technology Division
Code 5200
San Diego, California 92152
1 Copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, Maryland 20084
1 Copy

Captain Grace M. Hopper
NAICOM/MIS Planning Branch (OP-916D)
Office of Chief of Naval Operations
Washington, D.C. 20350
1 Copy

Advanced Research Projects Agency
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209
1 Copy

Dr. Stuart L. Brodsky
Office of Naval Research
Code 432
Arlington, Virginia 22217
1 Copy